LUNDS
UNIVERSITET
Lunds Tekniska Högskola

Palpable computing in a health care environment

TOMAS BJERRE

Master's Thesis at Lund Institute of Technology
Supervisor: Boris Magnusson and Emma Nilsson-Nyman
Examiner: Boris Magnusson

May 25, 2009

Abstract

Mark Weiser first coined the phrase “ubiquitous computing”. In ubiquitous computing computers exists everywhere and are not necessarily visible, meaning humans may use them without being aware of it. This scenario will require devices to adapt a flexible communication technique. A real scenario involving ubiquitous computing has been investigated. Different networking technologies has been looked into and conclusions has been made. Palpable computing has been chosen as foundation of a prototype system aiming at solving these problems. The PalCom framework has been used and evaluated as well as the developed prototype.

Acknowledgements

I would like to thank the people at LTH for making this thesis possible. Boris Magnusson for letting me work on this thesis and for his valuable feedback and ideas. Emma Nilsson-Nyman for her valuable feedback on the report. David Svensson for helping with PalCom related issues.

I would also like to thank the people at Medicinsk Teknik, Christian Reslow and Patrik Andersson. Medicinsk Teknik has been very helpful with providing equipment and answering questions.

Last but not least I would like to thank Shahram Khorsand, Sales Manager at Cardinal Health Sweden AB. He has been providing technical documents about the equipment used in this thesis.

Contents

List of Figures

List of Tables

1 Introduction **1**
- 1.1 Background . . . 1
- 1.2 Scenario - Service . . . 2
- 1.3 Scenario - Online usage . . . 3
- 1.4 Approach . . . 3
- 1.5 Outline . . . 4

2 Pervasive computing **5**
- 2.1 Introduction . . . 5
- 2.2 Interoperability in health care environments . . . 6
- 2.3 Interoperability in common network technologies . . . 7
  - 2.3.1 UPnP . . . 7
  - 2.3.2 Zeroconfig . . . 8
  - 2.3.3 Bluetooth . . . 8
  - 2.3.4 Jini . . . 8
  - 2.3.5 Conclusions . . . 9
- 2.4 Pervasive computing architectures . . . 9
  - 2.4.1 Obje . . . 10
  - 2.4.2 PalCom . . . 10
  - 2.4.3 Conclusions . . . 11
- 2.5 Conclusions . . . 11

3 Palpable computing **13**
- 3.1 Introduction . . . 13
- 3.2 Development strategy . . . 14
- 3.3 A user scenario: GeoTagger . . . 14
- 3.4 Architecture . . . 15
  - 3.4.1 Device . . . 15
  - 3.4.2 Service . . . 15
  - 3.4.3 Assembly . . . 16

3.4.4 Discovery . . . 16
3.5 The PalCom reference architecture . . . 17
3.6 Conclusions . . . 18

4 Service scenario prototype system **19**
4.1 Introduction . . . 19
4.2 PalCom design . . . 21
4.2.1 Service station interface . . . 21
4.2.2 Parser service interface . . . 21
4.2.3 Storage service interface . . . 22
4.2.4 Assembly script . . . 23
4.2.5 Data flow . . . 24
4.3 Service station service implementation . . . 25
4.4 Parser service implementation . . . 25
4.4.1 Algorithms . . . 26
4.4.2 Normalized log format . . . 27
4.5 Storage service implementation . . . 27
4.6 Web application . . . 28
4.6.1 Database interaction . . . 28
4.6.2 Graph generation . . . 29
4.7 Retrieving log data . . . 30
4.8 Conclusions . . . 30

5 Online scenario prototype system **31**
5.1 Introduction . . . 31
5.2 XSLT interface . . . 32
5.3 PalCom design . . . 33
5.3.1 Status information . . . 34
5.3.2 Alarm . . . 35
5.4 Conclusions . . . 35

6 Software distribution **37**
6.1 Distribution package . . . 37
6.2 Build process . . . 38
6.3 Implementing support for additional pumps . . . 38

7 Conclusions **39**
7.1 Introduction . . . 39
7.2 Working in a health care environment . . . 39
7.3 PalBox modifications . . . 40
7.4 PalCom evaluation . . . 40
7.5 PalCom versus a traditional approach . . . 42
7.6 Future work . . . 43

Appendices **43**

A Installation manual **45**
A.1 Web application 45
A.2 PalCom services 46

B User manual **49**
B.1 Downloading log data 49
B.2 Service station 50
B.3 Web application 51

C Portal online surveillance manual **55**
C.1 Preparing the PC 55
C.2 Configuring the Portal software 55
C.3 Using the Portal software 55

Bibliography **57**

List of Figures

2.1 Xerox devices: The ParcPad, the ParcTab and the LiveBoard 6

3.1 PalCom development process 14

4.1 Offline system overview. 20
4.2 PalCom services overview. 20
4.3 PalCom services data flow. 24
4.4 Service station GUI. 26
4.5 Web application - maximum number of simultaneously used devices. . . 29
4.6 Web application - time line. 29

5.1 Online system overview. 31

A.1 The PalBox device. 47
A.2 Unbound service in the PalBox device. 48
A.3 Assembly in the PalBox device. 48

B.1 Download software - main GUI. 50
B.2 Service station GUI. 51

B.3 Web application. 52
B.4 Web application - drug usage. 52
B.5 Web application - device models. 53
B.6 Web application - number of different devices. 53
B.7 Web application - maximum number of simultaneously used devices. . . 53
B.8 Web application - time line. 54
B.9 Web application - pump event. 54

List of Tables

4.1 Database interaction through the web application 28

Chapter 1

Introduction

The purpose of this chapter is to give an introduction to the report. Section 1.1 briefly discusses the general problems with electronic devices in health care environments. Section 1.2 and 1.3 explains the two main problem areas that this thesis focuses on. Section 1.4 describes the general approach to the problems and finally the outline of the entire report is presented in Section 1.5.

1.1 Background

There are software systems that operate on a single computer and there are distributed systems[3] that are distributed over several machines. Distributed systems consist of components that work together to reach a common goal.

At the University Hospital in Lund, just like any other hospital, there are many complex distributed systems in use. Health care today depends on many electronic devices such as X-ray machines and also pure software systems like record keeping systems. These systems rarely share communication abilities or agree on any standard of communication discipline. They use different physical interface for communication (like UART, Ethernet, Blue tooth, USB etc. or sometimes there own unique interface). They use different protocols for communicating. The actual data format of content in a response from a device also differ.

This thesis was initiated by Boris Magnusson at LTH (Lund Institute of Technology) and Patrik Andersson at MT (Medicinsk Teknik, medical technology division of the University Hospital in Lund). The University Hospital in Lund (USIL) has approximately 20 to 30 000 different electronic medical devices. Many of them maintain a local log of what is happening with the device. These devices can be for example medical pumps or life-support systems.

The IT systems at USIL have been growing more and more complex as new equipment has been added to the environment. The result being a very heterogeneous environment with many different software applications, interaction protocols and hardware. There are fourteen patient record systems[5]. This thesis focus on medical pumps that are used for continuous infusion therapy. These pumps contain

valuable information about usage that MT wants to use and make statistics of. MT has approximately 1200 such medical pumps in use today.

Lund Institute of Technology has been researching a methodology for ad-hoc composition of pervasive services. A methodology called palpable computing (see Chapter 3) has been developed as well as a framework for palpable computing called PalCom[5]. To use this framework in a real situation at the hospital is a good way of evaluating the technology.

1.2 Scenario - Service

The pumps are continuously brought in for service. For example if they do not work or just for a routine check. While at service MT wants to take care of the log existing on the device. These logs contain valuable information about usage. The fact that the interfaces and data format differ makes this a time consuming task.

The pumps are very expensive (around $3000 each) so MT is very interested in knowing how many pumps are actually used at a specific division. How many different pumps are ever used and how many pumps are used simultaneously at a specific division. If information can be extracted from the pumps it is also of interest to know what drugs has been used and what amount. Statistics on drug usage can be valuable input to decisions of what drugs need to be bought.

A few problem areas have been identified.

- **Download log files**
  To download the log file from the device. Since there are many different physical interfaces and communication protocols used, downloading data can be a complex process. Today this is done using software provided by each manufacturer.

- **Analyse log files**
  The manufacturers do not agree on a data format to use in the log files so to analyse them and get information about usage can be hard. The main problem here involves parsing out information and understanding the content of the log file.

- **Merge log files**
  The information from the log files needs to be translated into a normalized format. The information also needs to be merged with existing data from that same device and gathered in a collection among other devices.

- **Presentation of usage**
  The collected statistics from the pumps needs to be made available in a user friendly way.

1.3 Scenario - Online usage

A typical usage of a pump can be as follows.

1. A nurse programs the pump with what drug to use, the amount that will be pumped and the infusion rate.

2. He or she makes a note of this information.

3. He or she walks to a computer running a medical record system to enter the information.

In this case the information has to be formalised once at the pump, once on a note and once in the medical record system.

When an infusion comes to an end the syringe may need to be replaced with a new one and the infusion continued. It would be beneficial to issue an alarm whenever the infusion is close to finished. There are other things that can trigger such alarms. For example if the electric cable is suddenly unplugged.

A few problem areas have been identified.

- **Medical record system integration**
  Getting the information into the medical record system without having to make any notes.

- **Online surveillance**
  Enabling monitoring of alarms issued by a device.

1.4 Approach

The project was initiated by a small study of the hardware (the pumps). After a quick look at the pumps and discussions with people at LTH and MT the decision was taken not to focus on replacing the current download software (the software from each manufacturer used to download log data from the pumps). The pumps have very specialized communication disciplines. No documentation is available on the communication protocols or the interface hardware. Reverse engineering was considered but since there are legal issues invloved, that idea was discarded. Focus has been on the problem of having a collection of log files, with different content and format, that needs to be analysed and merged.

Pervasive computing (as described in Chapter 2) has been studied. Techniques related to pervasive computing has been studied, Palpable computing being one of them (as described in Chapter 3). Palpable computing was used in the prototype (see Section 4).

1.5 Outline

- **Chapter 1**
  Explains why this thesis was initiated and the main problems involved. It also describes the methodological approach used to find a fitting solution.
- **Chapter 2**
  Describes pervasive computing. The chapter discuss how it is dealt with in common communication techniques today. The chapter also presents a brief survey of identified techniques aimed at pervasive computing.
- **Chapter 3**
  Describes palpable computing and the PalCom framework.
- **Chapter 4**
  Presents the prototype software developed to deal with the service scenario.
- **Chapter 5**
  Presents the prototype software developed to deal with the online scenario.
- **Chapter 6**
  Presents the distribution package containing the developed software.
- **Chapter 7**
  Summarizes conclusions and potential future work.
- **Appendix A**
  Contains an end user manual on how to use the developed software for the service scenario.
- **Appendix B**
  Contains an end user manual on how to install the developed software for the service scenario.
- **Appendix C**
  Contains an end user manual on how to set up and use the developed software for the online scenario.

Chapter 2

Pervasive computing

The purpose of this chapter is to present pervasive computing and to discuss existing technologies dealing with the problems involved.

Pervasive computing is presented in Section 2.1. The interoperability problems involved in health care environments are discussed in Section 2.2. The traditional approaches to interoperability, standardisation, is discussed, along with different technologies practicing them, in Section 2.2. Two technologies specifically aimed at pervasive computing are discussed in Section 2.4.

2.1 Introduction

Pervasive computing is a post-desktop model of human-computer interaction presented by Mark Weiser[4]. In his vision information processing has been thoroughly integrated into everyday objects and activities. In the desktop-paradigm a user uses a single device for a specialized purpose, while in pervasive computing the user uses several devices simultaneously, and may not necessarily be aware of it.

Mark Weiser argues that whenever people learn something sufficiently well they cease to be aware of it. He gives an example with street signs and he says "When you look at a street sign you absorb its information without consciously performing the act of reading". There are many names for this phenomenon (compiling, tacit dimension, visual invariants, horizon, ready-to-hand, periphery)[4], but in all essence they all say the same thing. Only when things disappear in this way (as with street signs) we are freed to use them without thinking and to focus on new goals.

The motor is another example of something that has become very widely used and is now invisible. In the beginning of the 19th century there was typically a single engine that drove hundreds of different machines. Cheap, small, efficient electric motors made it possible first to give each tool its own source of motive force, then to put many motors into a single machine. In a modern car the driver uses many different motors without thinking about it. For example, for cleaning the wind shield or locking the doors.

Pervasive computing is basically about invisible computers interacting in a pervasive network. When Mark Weiser first coined the phrase he predicted that in the future there will be computers everywhere and as a natural part of our everyday life as books or road signs. This situation fits well with the situation at MT today. At MT as well as in any health care environment, the recovery, and even lives, of people depend on the stability of IT systems.

Figure 2.1. Xerox devices: The ParcPad, the ParcTab and the LiveBoard

Xerox PARC started a research project on pervasive computing in 1988. Other parts of Xerox PARC helped with inspiration of devices and functionalities. The LiveBoard, a wall sized computerized whiteboard with drawing and networking ability was developed. Xerox also developed the ParcPad that looks alot like a tablet PC and the ParcTab that looks more like a Palmpilot.

2.2 Interoperability in health care environments

Health care is introduced to new devices all the time. Some of these devices have been designed to be able to interact with other devices if integrated correctly. There is a great need for plug-and-play[7] interoperability[6]. To be able to take a device out of the box and just plug it in at the lab site and then easily make it work with the surrounding devices would be beneficial.

With interoperability the staff in the health care unit can use devices to their full potential. Another advantage of interoperability is that it encourages innovation and competition. When different products can be combined, and the communication disciplines are not kept secret, other and maybe specialized companies can extend functionalities of existing devices. Without interoperability, hospitals are forced to turn to large vendors that provide suites of compatible devices. These large vendors may not specialize in any one area. Interoperability promotes competition, and competition encourages innovation and quality.

Intel is a major manufacturer of health care devices and initiated the Continua Health Alliance[11, 8] in 2006. Intel identified six major factors that affect the authority of an industrie to achieve interoperability[6].

- **Demand** The health care units have to understand the need of interoperability and demand interoperable products from the manufacturers.

- **Standards** There must be standards defining what interoperability means in the field.
- **Business conditions** Business conditions must encourage manufacturers to make their products interoperable. For example the health care units has to be convinced of the extra value in a product that can interoperate with its surroundings.
- **Guidelines** Guidelines must exist that make the standards easier for companies to understand.
- **Compliance** Independent testing has to verify compliance to standards.
- **Promotion** Interoperability must be actively promoted.

Continua will establish technical design guidelines to provide manufacturers with the information needed to develop products such as interoperable sensors etc. Products made under Continua Health Alliance guidelines will provide consumers with increased assurance of interoperability between devices. Continua is currently a work in progress.

2.3 Interoperability in common network technologies

This section will discuss how interoperability is achieved in common network technologies. In these networks very little is known about the usage (compared to the scenario in Section 2.2). Traditionally interoperability between services in networks is achieved with standardisation. These standards are defined specifically for the application area where the services are used. The purpose of this section is to present a few common network technologies and to discuss how this standardisation is carried out.

2.3.1 UPnP

Universal Plug and Play (UPnP) is a set of computer protocols maintained by the UPnP Forum[16]. UPnP aims at simplifying the implementation of networks in the home and in corporate environments. DLNA[15] is built on top of UPnP and is a technology for connecting networked home audio/video equipment. It adds some well defined device types.

UPnP has a discovery mechanism but it is unable to handle discovery and communication over different kinds of networks. There is no configurable heart beat frequency that continuously search for and discovers new services. While there is no such heart beat the removal of services is done when they are needed, but not available.

Experts develop standards for devices and their services. There are, for example, standards for printers and scanners. The standards are specified using XML

Schema[1]. Different syntactical conventions are used in different device types[19]. This means that interaction with UPnP devices always have to be done after studying a written specification.

2.3.2 Zeroconfig

Zero Configuration Networking (Zeroconfig) is maintained by Apple and is a set of techniques that automatically creates a usable IP network without configuration or special servers [18]. This allows non-expert users to connect computers, networked printers, and other network devices and expect a functioning network to be established automatically.

Zeroconfig can choose a numeric network address for networked devices. It can automatically resolve and distribute computer host names. Zeroconfig has a discovery mechanism but it can not handle discovery and communication over different kinds of networks. Zeroconfig lacks the ability to report when devices are unavailable so services are removed when they are needed, but not available.

In Zeroconfig there is an informal process where anyone can add a service protocol to the list of registered protocols. This process has resulted in hundreds of standards[17]. Adding a service is as easy as sending an email with some short information.

2.3.3 Bluetooth

Bluetooth[25] is maintained by Bluetooth Special Interest Group[26]. Bluetooth was originally conceived as a wireless alternative to RS232 data cables. It can connect several devices.

Any device may perform an inquiry to find other devices to connect to, and any device can be configured to respond to such inquiries. If the device trying to connect knows the address of the device it always responds to direct connection requests and transmits device name and a list of services (among other things) if requested.

In Bluetooth several profiles are specified. These profiles contain protocols and procedures that devices must follow in order to be considered profile compliant.

Each Blutooth device has their own permanent and unique 48-bit address.

Bluetooth utilizes SDP (Service Discovery Protocol) to allow devices to discover what services other devices support. For example a mobile phone connecting to a Bluetooth headset will use SDP. SDP will determine which Bluetooth profiles are supported by the headset (for example Headset Profile).

2.3.4 Jini

Jini [13] was established in 1998 by Sun Microsystems[27]. Jini is a distributed computing environment that can offer network plug and play. Another device, or software service, can announce its presence and clients that wish to use such a

service can locate and invoke its services. Jini can be used in any network where there is some degree of change.

A digital camera can be connected to the network and present a user interface that will allow pictures to be taken. It may also be aware of any available printers that the pictures can be printed on.

Jini relies on a central repository of available services. The provider decides on a lease time, i.e., how long time will pass before an unavailable device is removed from the central repository.

A client obtains a proxy object from the service and invokes operations on the service which are performed via the proxy object. This means the client has to know exactly what the interface looks like. Both on a syntactical level and a semantic level. This means there is a need for standards for service types. The Jini community has started to standardise services and this movement has resulted in a standard for printers[14].

2.3.5 Conclusions

If a device has several interfaces that same device can have several addresses. These addresses may be dynamic and change from time to time. This will not work when we are only interested in the identity of the device.

Standards may seem very attractive. Many organizations count on the ability of the community to develop standards that guarantee seamless interoperability. While standards are useful, expectations on what can be achieved is unrealistic[9].

Standards can have drawbacks, e.g., they may be under specified. A domain standard needs to be exhaustive and include all possible functionality. Even small devices can result in complex standards[14]. Standards may also be conflicting, for example overlap another or compete with another standard. Standards may be inflexible. An early standard in a rapidly developed field can actually slow down development. Standards regarding health care tend to grow very large. Large standards may be hard to understand and admit to. Standards also needs to be updated and an updated version of a standard may not be compatible with an older version.

In a pervasive computing environment there are a lot of different devices. To standardise the communication of all these devices is a very time consuming task.

2.4 Pervasive computing architectures

There are many existing technologies that aim specifically at pervasive computing[28]. Obje and PalCom are presented in this section since these two are much alike and PalCom is used in a prototype in this thesis.

2.4.1 Obje

Obje[23] was formally known as SpeakEasy and was developed at Xerox PARC[22]. They seek to enable interoperability between devices without relying on domain-specific standards.

Obje has coined the term recombinant computing to characterize their architecture. The user can combine functionality from several services. The approach is to make configuration easy and to use and re-use in an ad-hoc manner.

Obje relies on distributed proxy objects with meta-interfaces consisting of mobile code. These proxies enables Obje to teach clients how to communicate. This mechanism allows one device to, among many things, teach another device how to render a secure media format or how to communicate with a transport protocol that guarantees a certain quality of service. It is possible to let users combine their clients with features that were unknown at the time the clients were developed. The focus in Obje has been on providing mechanisms for an end user without programming expertise. It is however not possible to combine services in any hierarchical manner. Obje is implemented in Java which means that each service needs to run in a JVM.

The development team of Obje has designed Obje with the consideration that new standards will appear each time a new type of device appears. The meta-interfaces abstract the protocols and communication standards used by specific devices so that a device can provide its communication specifics at runtime. This ability allows users to make decisions about which services they want to use and recombine them without waiting for vendors to agree on standards.

In the Obje framework, any discovery protocol can be represented as an aggregate component, which encapsulates collections of services discovered through the new protocol. Existing components can then communicate with these services via this bridging aggregate.

Obje has a user in the "loop of interaction", meaning that the user should always be in control when connecting services, leaving it up to him or her to make sure connections make sense. Users are expected to sort out much of the semantics, by determining when and why components should interact.

2.4.2 PalCom

PalCom is an architecture for pervasive systems, designed for ad-hoc compositions and non-preplanned interaction.

The service descriptions in PalCom are distributed as XML which can be handled on almost any device. These descriptions are free from any domain-specific operations. The operations on a service can be invoked by an assembly script or via a user interface. Each command on a service has a textual description enabling descriptive user interfaces to be automatically rendered from the service description.

PalCom focuses on providing mechanisms for an end user without programming experience. PalCom also has a focus on building assemblies where the control part

of a service is used as an API. The same interface description can be used both directly as a user and programmatic via an API.

The PalCom framework comes with a browser software that supports exploration of devices and services. Assemblies can also be constructed using this utility. It allows a user to explore new services and combine them in new or existing assemblies. The browser can also be used for inspection when things do not work. For example a running assembly can be inspected to find if all its connections are established. The browser can also connect to a specific service and provide a graphical user interface where the user can send and receive commands from the service.

Assemblies in PalCom can offer new services, called synthesized services. Synthesized services can then be used in other assemblies to form hierarchical constructions.

2.4.3 Conclusions

Both of these architectures seems promising. A few key differences are listed here.

- While Obje uses object code for communication, resulting in a relatively large over head, PalCom uses lightweight XML.
- Obje requires a JVM to run while PalCom can run on any device. For example an implementation has been done on an Axis camera[20].
- PalCom supports hierarchical structures of combined services.

2.5 Conclusions

This chapter has presented pervasive computing and discussed some of the problems involved. Standardisation approaches where discussed in Section 2.3 with the conclusions that standards are inflexible and time consuming. Two architectures specifically aimed at pervasive systems were presented in Section 2.4 concluding that alternatives exists and that PalCom seems most promising.

Chapter 3

Palpable computing

This chapter will introduce palpable computing in Section 3.1. The strategy used for developing the architecture is discussed in Section 3.2. There are many examples of user scenarios used to take the project forward, one of them is presented in Section 3.3. The general architecture and concepts of palpable computing used in this thesis is presented in Section 3.4. A bit more technical description of the relevant parts of PalCom is presented in Section 3.5. The chapter is concluded in Section 3.6.

There is a doctoral thesis "Assemblies of Pervasive Services"[19], by David Svensson Fors, that presents palpable computing and the PalCom framework in detail. There is also a deliverable document and a developer companion document on the PalCom project web page[24].

3.1 Introduction

Palpable computing is about designing IT that is easy to understand and modify for users. It is about "doing pervasive computing right". Palpable computing lets the technology be visible when it has to be and understandable all the time. Palpable computing lets the technology be noticeable and understandable instead of invisible. Users have enough control over the device to, for instance, inspect and correct errors.

Palpable devices should be able to handle changes in their immediate environment. As a new device is added or an existing device is removed, or replaced by a new device, the other devices can sense this and understand what happened. To minimise the necessary decisions made by the user, the device can choose in what way to connect. The user always have the ability to make the final decision.

Palpable computing is an invention by scientists and professionals from, the European research project, PalCom. These people are developing a design concept for palpable computing. The concept includes an open software architecture, a toolbox with software components and design tutorials. About a hundred people are involved in the development and are working with many different aspects of palpable computing. A list of PalCom partners can be found at the project webpage[1].

[1] http://www.ist-palcom.org/consortium/partners-and-people/

3.2 Development strategy

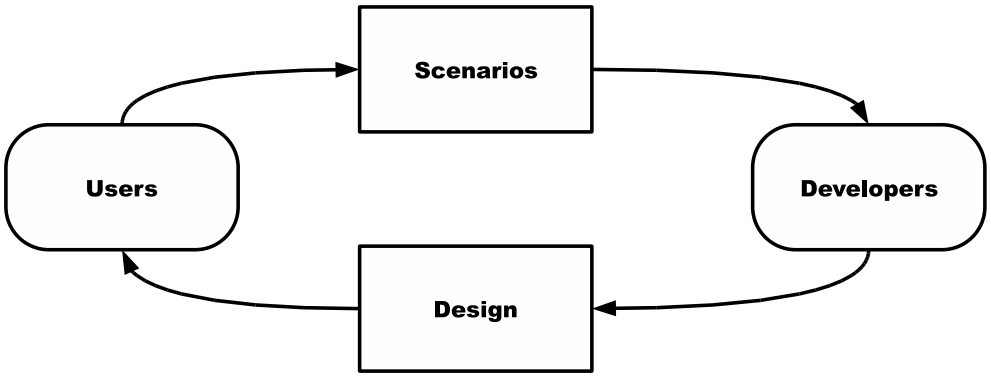

Figure 3.1. PalCom development process

The design process is based on real life scenarios (see Figure 3.1). Researchers study how users, from the projects application areas, make use of current and emerging technologies. The need for a certain type of device interaction takes the project forward.

3.3 A user scenario: GeoTagger

This section will present a user scenario called GeoTagger[19, 5]. The purpose is to show how palpable computing can be used. This is the kind of scenario that have been used in the development strategy, as mentioned in Section 3.2.

The GeoTagger scenario originates from landscape architects in Scotland. The landscape architects analyse the surrounding environment of a planned building. They need to take a lot of pictures of the environment and pair them with coordinates from a GPS as well as direction from a compass. It would be beneficial to automate this process and to send the collected information to a central storage.

Assuming that the devices are "palpable", this scenario can, for example, be solved as follows.

The devices to be used (compass, GPS and camera as well as an additional Storage service) is connected with there respective interfaces. The storage is a central storage service.

Using the PalCom developer browser, from the PalCom distribution, an assembly (see Section 3.4.3) can be developed. The assembly can be developed using an XML editor or using the graphical editor in the browser. All palpable devices are automatically discovered (see Section 3.4.4) and will show up in the browser. The service descriptions (see Section 3.4.2) of each device is inspected to get an idea about the semantics. Two assemblies are created, Uploader and TaggingCamera.

TaggingCamera is constructed using the Camera, the GPS, the compass and an unbounded service (see Section 3.4.2) called CoordinateStuffer from the PalCom distribution. When a picture is taken with the camera it will send the picture to TaggingCamera which will ask the GPS for a coordinate and the compass for direction. The coordinate, the direction and the picture is sent to CoordinateStuffer which will respond with a picture including coordinates and direction as meta data. This response is provided in a synthesized service (see Section 3.4.2) to make the picture available to other assemblies.

Uploader will react on a new picture being sent from the synthesized service of the TaggingCamera assembly and transmit that picture to the central storage service.

The landscape architecture can now bring his devices out in the field and automatically transmit every picture taken with the camera. The assemblies can run on a handheld device, for example a cell phone. A tunnel, over GPRS, will have to be set up between the cell phone and the office in order for the Uploader service to function. The devices will automatically be connected by the assembly manager on the cell phone.

3.4 Architecture

This is an ongoing project and this architecture may change. This is a short summary including relevant parts for this report. A much more detailed description can be found in the latest deliverable of Open Architecture (54 was the latest when this report was written).

3.4.1 Device

Devices are physical or virtual with which users interact with palpable systems. They host one or more runtime environments and/or one or more Assemblies.

Devices are identified by a DeviceID. The DeviceID does not change over time and is unique to the device rather then unique to an interface (like an IP address on an TCP/IP interface). The DeviceID can be assigned by the user, derived from the MAC address of any Ethernet hardware or a UUID (Universally Unique Identifier). The devices also have another non unique name that is more describing. This name is used when presenting the device to a human being.

3.4.2 Service

A Service is an interaction point of a Device. The communication is always between Services. The service description is expressed in XML.

Control Services are used in this thesis and are Services that has a set of input and/or output commands. The commands may also contain one or more parameters where each parameter is identified by a name and has its own MIME type.

Stream Services sends or receives a stream of data. Stream services has not been used in this thesis.

There are also meta services that does not provide any domain functionality. A meta service can for example render a user interface from a service description. Meta services has been used for debugging during development in this thesis.

3.4.3 Assembly

An Assembly is a collection of resources organized to deliver all or part of some application functionality. It is the element in PalCom that enables services to be combined with other services. The assembly needs to contain at least a list of devices, a list of services and a list of connections. An assembly description is expressed in XML.

It is possible to add coordination to the assembly using an assembly script. The scripting feature enables the assembly to handle incoming commands from any service. The assembly can pass command parameters from an incoming command to any other command on any service or store parameter values in temporary variables in the assembly itself.

The assembly can create synthesized services where functionality gathered from other services are exposed as a service. Synthesized services can be used to create hierarchies of assemblies.

One idea behind the scripting language has been to keep it simple enough for an end user, without any scripting experience, to understand and use it. A concept of unbounded services (see Section 3.4.2) has been developed to enable more advanced functionality but still keep the scripting language simple.

The assembly is executed by an assembly manager. The assembly manager establishes the needed connections in the assembly. If there is an assembly script it will be interpreted and the assembly manager will provide its functionality.

3.4.4 Discovery

The discovery mechanism is a weak part in many existing network technologies2.3. This section will discuss how this is dealt with in palpable computing.

The discovery mechanism is used by devices to announce and use services. PalCom is designed to be used in local networks but it is possible to connect small networks with tunneling [21].

A browser software, such as PalCom Developer Browser from the PalCom distribution, discovers surrounding devices and enables interaction with them. A good example of the non pre planned interaction philosophy behind PalCom is how such a browser can automatically generate a user interface from any discovered service.

A heartbeat mechanism is used to discover changes in the surrounding. Each device can, optionally, broadcast themselves using a HeartBeat message containing their address. Any device that is online has to respond with a HeartBeatAck message containing their address in order for them to be considered available. The

HeartBeat messages are sent periodically with a time delay decided by the device with the shortest period. Any device will send a HeartBeat if there has been no HeartBeats during that devices shortest period.

A device can be considered to be unavailable for two reasons. If no HeartBeatAck or HeartBeat message has been received during a period of time. Or the device sends a HeartAttack message. A HeartAttack message means the device sending it will go offline and tells every other device to remove its information.

The heartbeat mechanism can potentially lead to a lot of data being sent in every heartbeat message. Whenever a service description has changed all other devices needs to know about it and sending all service descriptions in all heartbeat would result in unnecessary bandwidth usage. This situation is resolved by sending cache values, that are only changed when the descriptions are changed, instead of the descriptions.

Small devices may only need to be discoverable and may not care about what other devices are available in their surrounding area. These small devices have the option of not sending any HeartBeat messages but only respond to them using HeartBeatAck.

3.5 The PalCom reference architecture

PalCom is currently the only implementation of palpable computing and serves as a reference architecture. The main part of the implementation has been done at the Computer Science department of Lund Institute of Technology.

PalCom is open source. The entire source code can be found in a Subversion repository[2].

PalCom has been designed on top of a virtual machine (VM). A virtual machine called Pal-VM has been developed to deal with resource constrained devices and can run a subset of Java called Pal-J. A PalCom Device can also run on a JVM (Java Virtual Machine). Pal-J does not support threads the same way as ordinary JVM. Pal-J is the reason why PalCom uses its own kind of thread called PalComThread. PalComThreads are based on coroutines. These threads do not interrupt each other and only one thread can run simultaneously. Each thread decides when it is done so that another thread can start executing.

In the PalCom distribution there is an abstract class called AbstractDevice. AbstractDevice has a run method that schedules middleware managers and services in a PalcomThread. AbstractDevice provides functionality for such things as setting DeviceID and initiate and add services to the device. A device is implemented by sub classing this abstract class.

There is also an abstract class called AbstractService that handles announcement of a service. A concrete instance of this class can define commands. The service uses a PalComThread for sending and receiving commands.

[2]http://svn.daimi.au.dk/svn/palcom/

3.6 Conclusions

Palpable computing has been presented as well as the concepts relevant for the prototype system (see Chapter 4).

Chapter 4

Service scenario prototype system

The purpose of this chapter is to present the prototype system developed to deal with the service scenario described in Section 1.2. This chapter will present the technical aspects of the system. An end user manual is available in Appendix A and B.

There were some initial considerations of different approaches which are described in Section 4.1. Section 4.1 also provides an overview of the system design.

Section 4.3, 4.4 and 4.5 will describe the PalCom services present in the system design. Section 4.2.4 will present the assembly along with the assembly script that binds the services together to form the requested functionality. A use case is also available to show the information flow in the system.

Conclusions are gathered in Section 4.8.

4.1 Introduction

To kick start the project an early and very basic solution was developed. The goal of this first iteration was to see what was possible to do with a traditional approach. With this knowledge a more flexible solution could later be developed.

The first iteration consisted of a Java application that read and parsed a log file. The log was retrieved using applications provided by the manufacturer of the device. The resulting parsed log information was sent to a web server via POST. The web server stores the information in a database. A web application was developed that renders the database and present its content graphically. This approach allowed for simultaneous development of different parts of the system.

Considering the first iteration, four different PalCom services were suggested.

- **Storage service**. A service that would provide a persistent storage where parsed log information could be stored.

- **Parsing service**. A service that would handle parsing of raw log data and provide information in a normalized way.

- **Service Station service**. A service that would enable an end user to interact with the system in a user friendly way.
- **Pump service**. A service that would, theoretically, run on a specific pump.

The Pump service was later discarded as the pump devices are hard to interaction with due to lack of documentation. The idea was that this service would publish information from each pump on the network. The service station would use these services to gather log data and pass it through to the parser service upon request by the end user. If a development environment was available, where software could be developed to run on the pump, this kind of service would have been a good idea. Instead raw log files are gathered with existing tools and the service station detects these log files and uses them as data source instead.

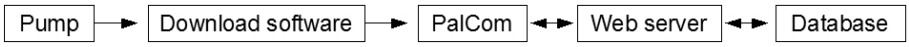

Figure 4.1. Offline system overview.

Figure 4.1 shows the data flow in the implemented system described in this chapter. The log data is extracted from the pump device using vendor software. PalCom services are used for parsing and storing as well as publishing the log data. PalCom uses a web application for persistant storage and lookup of information. The web application also provides the end user of the system with usage statistics of pump devices. The web application provides this functionality by taking advantage of a MySQL database.

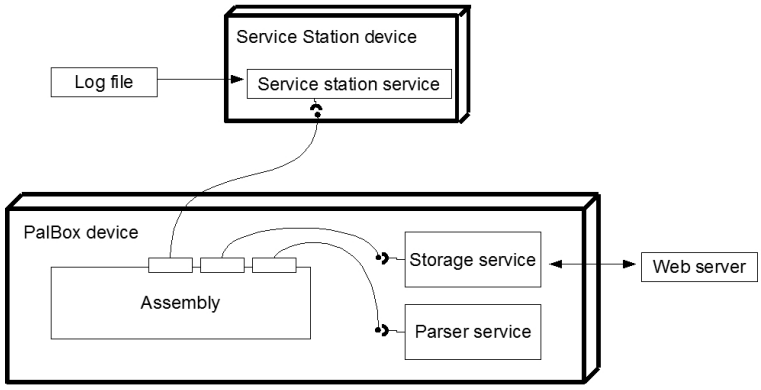

Figure 4.2. PalCom services overview.

The final PalCom design is shown in Figure 4.2. The parser and storage services are running as unbound services (see Section 3.4.2). The assembly is running on a modified version of the PalBox device found in the PalCom distribution (see Section 7.3).

4.2 PalCom design

This section will describe how PalCom has been used in this scenario.

4.2.1 Service station interface

```
<SD id="ServiceStationService" ls="1">
  <CmdI id="ParseAndStoreLog" direction="out">
    <PI id="InventoryNumber" type="text/plain" dataRef="0" />
    <PI id="Log" type="text/plain" dataRef="0" />
  </CmdI>
  <CmdI id="StatusResponseError" direction="in">
    <PI id="InventoryNumber" type="text/plain" dataRef="0" />
    <PI id="Status" type="text/plain" dataRef="0" />
  </CmdI>
  <CmdI id="StatusResponse" direction="in">
    <PI id="InventoryNumber" type="text/plain" dataRef="0" />
    <PI id="Status" type="text/plain" dataRef="0" />
  </CmdI>
  <CmdI id="StatusResponseCompleted" direction="in">
    <PI id="InventoryNumber" type="text/plain" dataRef="0" />
  </CmdI>
</SD>
```

Listing 4.1. Service station service interface

The service interface of the Service station will only have one output command, ParseAndStoreLog, as can be seen in Listing 4.1 ParseAndStoreLog will publish the inventory number of the pump along with the raw data retrieved from the pump. The service also has three different input commands. The input commands are used by the other services to report the status of the published pump device.

- **StatusResponse** will simply set the status information in the GUI (see Figure 4.4).
- **StatusResponseError** will report that an error has occurred and set the status information to an appropriate error message. This functionality allows, for example, the parser software to decide on the error message instead of the actual service station. A system of error codes and translations into actual status information is avoided.
- **StatusResponseCompleted** will tell the Service station that a specific device log has been successfully parsed and stored.

4.2.2 Parser service interface

```
<SD id="ParserServiceProxy" ls="8">
  <CmdI id="PreLoad" direction="in">
    <PI id="InventoryNumber" type="text/plain" dataRef="0" />
    <PI id="Log" type="text/plain" dataRef="0" />
  </CmdI>
  <CmdI id="PreLoadResponse" direction="out">
```

```
7       <PI id="InventoryNumber" type="text/plain" dataRef="0" />
8     </CmdI>
9     <CmdI id="PreLoadResponseError" direction="out">
10      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
11      <PI id="Message" type="text/plain" dataRef="0" />
12    </CmdI>
13    <CmdI id="GetParsedLog" direction="in">
14      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
15      <PI id="Parser" type="text/plain" dataRef="0" />
16    </CmdI>
17    <CmdI id="GetParsedLogResponse" direction="out">
18      <PI id="ParsedLog" type="text/plain" dataRef="0" />
19      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
20    </CmdI>
21    <CmdI id="GetParsedLogResponseError" direction="out">
22      <PI id="Message" type="text/plain" dataRef="0" />
23      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
24    </CmdI>
25  </SD>
```

Listing 4.2. Parser service interface

The parser was originally designed with only three commands, one for requesting a parsed log given an algorithm and raw log data and two response commands for successful or erroneous parsing. The problem with that approach was concurrency. The raw log is provided by the Service station and can be directly directed to the parser. The second parameter, the algorithm has to be requested from the storage service and when a response is received, the parser service can be told to start parsing.

In this case the raw data has to be temporarily stored somewhere. The assembly does provide a temporary storage in variables. There is no support for concurrency and the variable will be overwritten if several parsing requests are performed simultaneously. This is the reason why there is a pre load functionality in the parser service.

The PreLoad command will take the raw log data and store it using the inventory number as key. The GetParsedLog command can then be invoked in order to use the specified algorithm for parsing the raw log data. Each input command has a couple of output commands, one for successful execution and one for erroneous.

4.2.3 Storage service interface

```
1   <SD id="StorageServiceProxy" ls="10">
2     <CmdI id="StoreParsedLog" direction="in">
3       <PI id="Log" type="text/plain" dataRef="0" />
4       <PI id="InventoryNumber" type="text/plain" dataRef="0" />
5     </CmdI>
6     <CmdI id="StoreParsedLogResponse" direction="out">
7       <PI id="InventoryNumber" type="text/plain" dataRef="0" />
8     </CmdI>
9     <CmdI id="StoreParsedLogResponseError" direction="out">
10      <PI id="Message" type="text/plain" dataRef="0" />
11      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
12    </CmdI>
13    <CmdI id="GetParser" direction="in">
```

```
14      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
15    </CmdI>
16    <CmdI id="GetParserResponse" direction="out">
17      <PI id="Parser" type="text/plain" dataRef="0" />
18      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
19    </CmdI>
20    <CmdI id="GetParserResponseError" direction="out">
21      <PI id="Message" type="text/plain" dataRef="0" />
22      <PI id="InventoryNumber" type="text/plain" dataRef="0" />
23    </CmdI>
24  </SD>
```

Listing 4.3. Storage service interface

The service interface, as presented in Listing 4.3, is closely related to the database API presented in Section 4.1. Each command will result in a request to the database and the response will be passed through back to the service.

The interface has been designed with the same general idea as the interface of the Service station and the Parser service. There are two input commands and both have two output commands associated with them, one for successful responses and one for erroneous. The GetParser command will request the parser algorithm associated with the given inventory number. The StoreParsedLog will request the given normalized XML log to be stored.

4.2.4 Assembly script

The assembly script used to bind the Parser, Storage and Service station together is presented in Listing 4.4. The flow of data between the services is described in Section 4.2.5.

```
1
2   when PreLoadResponse from ParserService
3       send StatusResponse(thisevent.InventoryNumber, 'Log pre loaded in parser
            .') to ServiceStationService
4       send GetParser(thisevent.InventoryNumber) to StorageService
5   when PreLoadResponseError from ParserService
6       send StatusResponseError(thisevent.InventoryNumber, thisevent.Message) to
             ServiceStationService
7   when GetParserResponseError from StorageService
8       send StatusResponseError(thisevent.InventoryNumber, thisevent.Message) to
             ServiceStationService
9   when GetParserResponse from StorageService
10      send StatusResponse(thisevent.InventoryNumber, 'Parser algorithm received
             from database.') to ServiceStationService
11      send GetParsedLog(thisevent.InventoryNumber, thisevent.Parser) to
            ParserService
12  when GetParsedLogResponseError from ParserService
13      send StatusResponseError(thisevent.InventoryNumber, thisevent.Message) to
             ServiceStationService
14  when GetParsedLogResponse from ParserService
15      send StatusResponse(thisevent.InventoryNumber, 'Log parsed, sending to
            storage.') to ServiceStationService
16      send StoreParsedLog(thisevent.ParsedLog, thisevent.InventoryNumber) to
            StorageService
17  when StoreParsedLogResponseError from StorageService
18      send StatusResponseError(thisevent.InventoryNumber, thisevent.Message) to
             ServiceStationService
```

```
19 when StoreParsedLogResponse from StorageService
20     send StatusResponseCompleted(thisevent.InventoryNumber) to
           ServiceStationService
21 when ParseAndStoreLog from ServiceStationService
22     send PreLoad(thisevent.InventoryNumber, thisevent.Log) to ParserService
23     send StatusResponse(thisevent.InventoryNumber, 'Pre loading log in parser
           .') to ServiceStationService
```

Listing 4.4. Assembly script

4.2.5 Data flow

The data flow of the services is described here using a typical use case. The end user has set up the Service station. He or she is about to press the send button of a device log in the user interface.

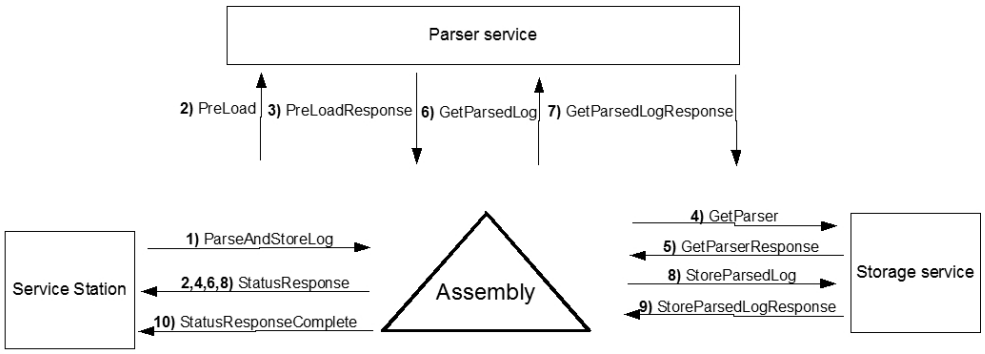

Figure 4.3. PalCom services data flow.

- **Send button pressed**
  The device log file is compressed using the gzip format and the ParseAndStoreLog command is invoked. The assembly script will respond to this command with a StatusResponse and it will also invoke the PreLoad command on the ParserService.

- **Parser pre loaded**
  The parser service will pre load the data given by the PreLoad command and respond with the PreLoadResponse command. The assembly will trigger on that response and update the status text in the service station using the StatusResponse command. It will also invoke the GetParser command on the storage service.

- **Storage parser algorithm lookup**
  The storage service will lookup the algorithm to be used and respond with the GetParserResponse command. The assembly will trigger on that response

and update the status text in the service station using the StatusResponse command. It will also invoke the GetParsedLog command on the parser service.

- **Parser algorithm execution**
  The parser service will execute the algorithm on the pre loaded log data. When the execution is finished it will invoke the GetParsedLogResponse command. The assembly will trigger on that response and update the status text in the service station using the StatusResponse command. It will also invoke the StoreParsedLog command on the storage service.

- **Final storage**
  The storage service will invoke the web application to store the normalized log. The StoreParsedLogResponse command is invoked. The assembly will trigger on this command and invoke the StatusResponseCompleted on the Service station. The send button in the user interface will turn green and the entire procedure is finished.

4.3 Service station service implementation

The Service station main functionality is to enable the end user to interact with the other PalCom services. The goal during development has been to, as easy as possible, publish log data via PalCom.

The Service station will look for files on the file system at a specific location. The filename determines the inventory of the pump from which the log data was retrieved. The inventory number is later used, along with the content of the file, to extract information.

The Service station can make one or several log files available simultaneously or put several log files in a queue that are made available one at a time.

As the usability of the entire system depends a lot on a user friendly interface for the end user, effort has been put into the design of the graphical user interface, GUI, see Figure 4.4. Red and green colors has been used to trigger the cognitive awareness of success or failure. In order to give the end user more control over the system it is up to that person to decide if a log file is to be considered as processed. The GUI will indicate if the parsing and storing was successful, or not and will then leave it up to the user to decide if the log file should be discarded.

4.4 Parser service implementation

The parser service is responsible for normalizing raw log data and make it the information available in a well known format. The service achieves this task by interacting with the storage service to get the correct algorithm to use for parsing every specific log data.

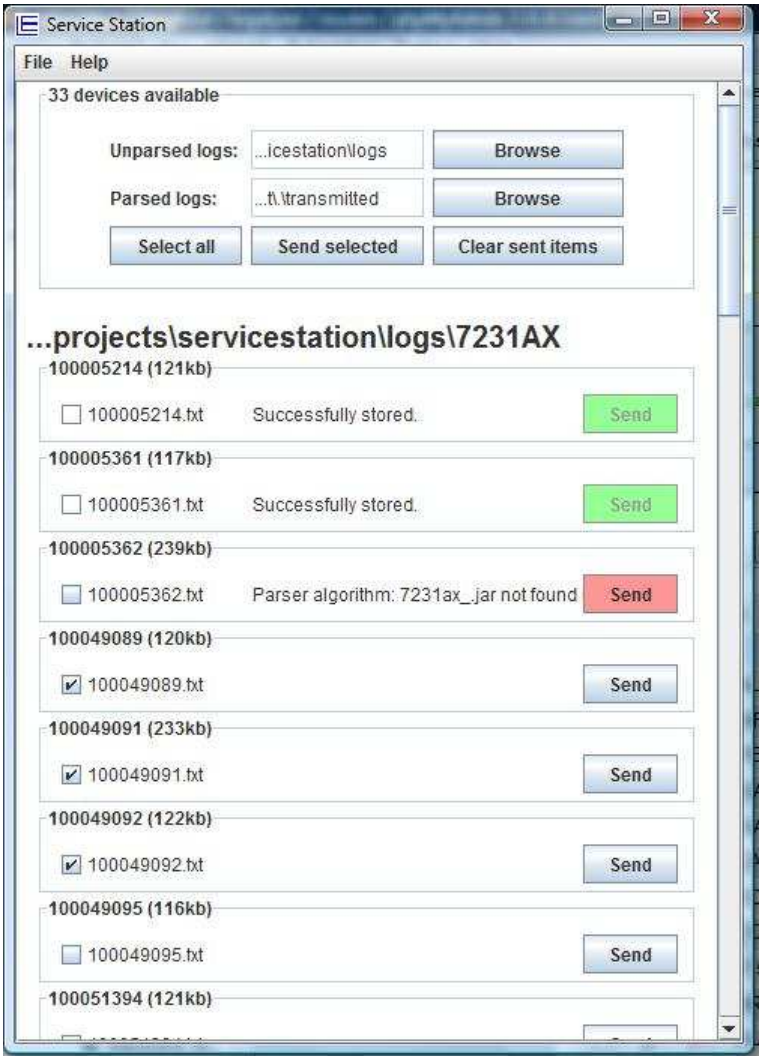

Figure 4.4. Service station GUI.

4.4.1 Algorithms

Every log format is different and there are no guaranties of consistency between models or between firmware versions. With this in mind it is obvious that there is a gain in separating the actual parsing algorithms from the rest of the parser service. The algorithms being the logics involved in extracting information from log files.

At first the algorithms was separated into Java packages with a strong dependency to the rest of the parser. With such a strong dependency the entire parser service had to be recompiled in order for it to be extended with a new parser algorithm. The need for a plug in system was obvious. The plug in system was originally

designed using executable JAR files as plug ins and executing them in a separate operating system thread. This approach resulted in weird bugs and there was a need for clean up process if the algorithm would hang. This approach was therefore discarded and a plug in system using dynamic loading of classes was developed.

The task that each algorithm is assigned is to extract information from raw log data and provide a normalized XML document containing usage information.

As algorithms were developed a framework for the algorithms was developed in parallel. The framework provides functionality that will make it easier to implement new device support. There is an abstract class that has to be extended to create an algorithm. The algorithm has to implement a method called *parse*. In that method the raw log data is available and the developer creating the algorithm will only have to populate a couple of data structures in order to create a fully functional parser. The framework then uses these data structures to generate XML.

A description of how to implement a new algorithm is provided in Section 6.3.

4.4.2 Normalized log format

The normalized log format is based on XML[1]. The entire log is encapsulated in an `eventlog` tag. The `eventlog` tag holds zero or more `event` tags. Each `event` tag represents a specific event. Each event is described with type, start and end timestamps, infused amount (in ml), what drug and drug library as well as what syringe was used. If any of the values are unavailable in the log they will be set to null. The type is currently always "operation", this tag is only available to provide forward compatibility. An example log is available in Listing 4.5.

```
1  <eventlog>
2   <event>
3    <type>operation</type>
4    <start>2008-12-22 10:40:24</start>
5    <end>2008-12-22 16:19:33</end>
6    <amount>19.68</amount>
7    <druglib>ANIVA  (v 1)</druglib>
8    <syringe>BD Plastipak 50ml</syringe>
9    <drug>Propo 20</drug>
10  </event>
11 </eventlog>
```

Listing 4.5. Normalized event log

4.5 Storage service implementation

The storage service is responsible for providing a persistent storage to the PalCom environment. It will interact with a web application to perform storing and lookup of information.

4.6 Web application

The web application has been developed using PHP and PHP-GD has been used to create graphs in real time.

Security issues has not been a priority during development of the system. The web application has been designed with a minimal login procedure in order to give a simple protection against viruses and spammers. A spam bot sending random requests could otherwise add unwanted statistics to the database. The login procedure also allows for security improvements in the future.

4.6.1 Database interaction

Task	*Input*	*Output*
Login	request, username, password	response, sessionid
Logoff	request	response
Get algorithm	request, inventorynbr, sessionid	response, parser
Store log	request, xmllog, sessionid	response, description

Table 4.1. Database interaction through the web application

Table 4.1 shows the parameters involved when interacting with the database through the web application. To login and get a session id, the request parameter should be set to "login". If the username and password is correct a session id is returned. The response parameter is set to "success". If username and password is invalid the response parameter is set to "failure".

To logoff, the request parameter should be set to "logoff". The response will always be "success".

To get a parser algorithm associated with a device the request parameter is set to "getparser". The session id retrieved from a login request will have to be provided in the sessionid parameter. The response parameter will either be set to "DeviceNotFound" or "Success", if successful the parser parameter is set to the name of the associated algorithm.

To send a normalized log to the database the request parameter is set to "sendlog". The `inventorynbr` parameter is set to the inventory number of the device. The `xmllog` parameter is set to the normalised event log. The session id retrieved from a login request has to be provided. If the event log contains unsupported event types the response will be set to "EventTypeUnknown" and the description parameter is set to the unsupported event type. The response parameter can also be set to "DeviceNotRegistered" meaning that is is not present in the database. If the storing process was successful the response parameter is set to "success".

4.6.2 Graph generation

To make the gathered statistics available and understandable, graphical representations has been developed. Most graphs are created with the PNG format using PHP-GD. The PNG format enables easy insertion into existing reports. An example of a graph is presented in Figure 4.5. The graph presented in Figure 4.5 was created using pure HTML formatting. All different graphs are further described in Section B.3.

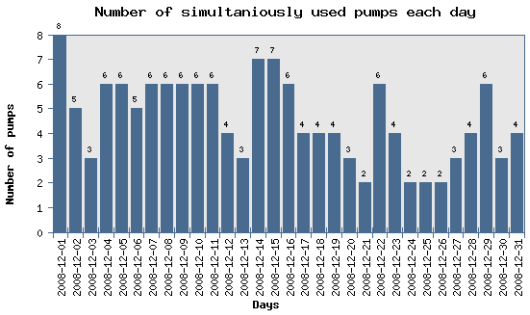

Figure 4.5. Web application - maximum number of simultaneously used devices.

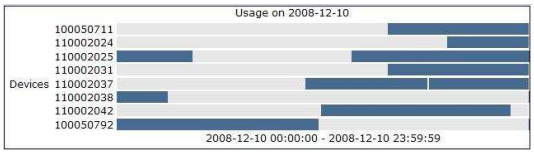

Figure 4.6. Web application - time line.

While most graphs can be implemented with almost only SQL, there is one graph in particular that offered more of a challenge. The task was to visualise the simultaneous usage of pumps given a time period. All event information as seen in Listing 4.5 is stored at each row in a table in the database. The task has been divided into three separate methods, *calcPumpsUsed*, *getUsageSim* and *getUsageSimData*. The implementation is found in `usage.inc.php` in the distribution package (see Section 6.1). The task has been separated into these three methods to make the code easier to test, by dividing responsibility and reduce the complexity.

The `calcPumpsUsed` will provide an array with devices as keys. The value is an array of time intervals within the given start and end time. Each time interval has start time as key and end time as value. Intervals that are only partly inside are shrunk before they are added to the array of intervals.

The `getUsageSim` will use the calculated intervals and create new intervals. The new intervals has a weight attached to them corresponding to the number of simultaneous devices during that interval.

The `getUsageSimData` method will calculate the maximum number of simultaneous devices used within the given resolution. Currently only simultaneous usage per day is requested by MT and therefore the only resolution implemented. This method will simply have to lookup the maximum value in the array given by `getUsageSimData`.

4.7 Retrieving log data

Log files have mainly been retrieved using software maintained by the pump device manufacturers. A few weeks after the project started, one of the device manufacturers changed there software so that log data could no longer be retrieved. The pump device, Alaris Asena Syringe Pump, has a COM port for serial communication. An application has been developed that will scan the serial ports of the computer to look for any connected device. The connected devices will appear in a GUI allowing the user to connect and download the log file. The software can be found in the LogExtractor project in the distribution package (see Section 6.1). A user guide is available in Section B.1.

This communication method turned out to be very slow. A pump containing 18000 events will need about 1 hour to transmit all events.

4.8 Conclusions

As PalCom itself can be considered a prototype framework it is not completely reliable. The services developed in this thesis may therefore lose there connection to the assembly sometimes. This is however very rare and every time it has happened the situation has been resolved with a restart of the services.

Chapter 5

Online scenario prototype system

The Alaris Gateway Workstation[1] has been investigated to explore the possibility of online surveillance. Pumps can be attached to this device and monitored via an IP network. The findings is discussed in this chapter.

An overview of the system is presented in Section 5.1. The communication between the PalCom service and the workstation is described in Section 5.2. The usage of PalCom is presented in Section 5.3. Conclusions are gathered in Section 5.4.

5.1 Introduction

Figure 5.1 shows an overview of the implementation. A PalCom service has been developed to provide information retrieved from the Alaris Workstation. The assembly and the alarm system illustrated in Figure 5.1 has been left for future work.

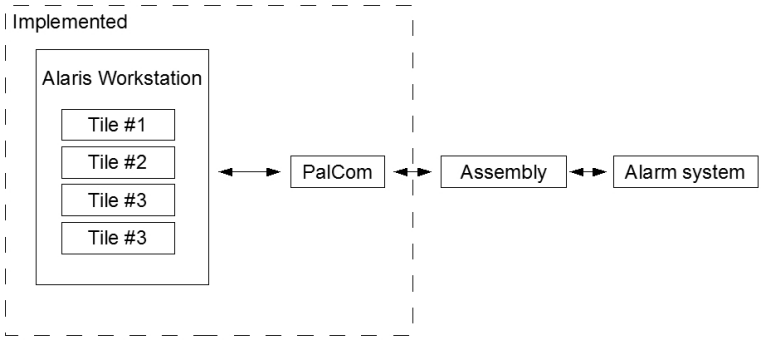

Figure 5.1. Online system overview.

[1] http://www.alarismed.co.uk/products/d/accessories/800203UN01-34/

5.2 XSLT interface

The workstation provides an XML Schema Language Transformation[2], XSLT, interface mechanism. An XSLT request is sent to the workstation containing a description of the requested information, see Listing 5.1. The workstation will respond with an XML document containing the actual requested information, see Listing 5.2. The communication socket can be kept open by the client and if so, the workstation will continue to respond whenever the requested content has changed.

```
1  <?xml version="1.0" encoding="UTF-8" ?>
2  <xsl:stylesheet version="1.0" xmlns:alaris="http://www.alarismed.com/XML/
       AsenaGateway.xsd" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
3  <xsl:output method="xml" indent="yes" version="1.0" encoding="UTF-8" />
4  <xsl:template match="/">
5  <Gateway>
6   <xsl:for-each select="alaris:Gateway">
7    <Tiles>
8    <xsl:for-each select="*/*/alaris:Pump">
9      <Tile>
10     <Pump>
11       <xsl:attribute name="serial_no">
12        <xsl:value-of select="@serial_no" />
13       </xsl:attribute>
14       <xsl:copy-of select="alaris:Infusion" />
15       <xsl:copy-of select="alaris:Patient" />
16     </Pump>
17     </Tile>
18   </xsl:for-each>
19   </Tiles>
20  </xsl:for-each>
21 </Gateway>
22 </xsl:template>
23 </xsl:stylesheet>
```

Listing 5.1. Example XSLT request

```
<?xml version="1.0" encoding="UTF-8"?>
<Gateway xmlns:alaris="http://www.alarismed.com/XML/AsenaGateway.xsd">
 <Tiles>
  <Tile>
   <Pump serial_no="8003-07204">
    <Infusion xmlns="http://www.alarismed.com/XML/AsenaGateway.xsd">
     <Rate volume_units="ml" time_units="h">9.00</Rate>
     <VolumeInfused units="ml">11.964</VolumeInfused>
     <Pressure>
     <Variable units="mmHg">0</Variable>
     </Pressure>
     <TimeRemaining>02:07:12</TimeRemaining>
     <Patient>
      <Weight units="kg">70.00</Weight>
     </Patient>
     <Syringe brand="ASTRAZENECA">
      <Size units="ml">50</Size>
     </Syringe>
    </Infusion>
   </Pump>
  </Tile>
 </Tiles>
</Gateway>
```

Listing 5.2. Example XSLT response

5.3 PalCom design

The PalCom interface developed provides status information about the workstation as well as the pumps attached to it. The PalCom interface also provides an alarm functionality.

```
<SD id="SurveillanceService" ls="1">
    <CmdI id="GetIdentity" direction="in" />
    <CmdI id="Identity" direction="out">
      <PI id="Serial" type="text/plain" dataRef="0" />
      <PI id="Ward" type="text/plain" dataRef="0" />
      <PI id="Bed" type="text/plain" dataRef="0" />
      <PI id="Hospital" type="text/plain" dataRef="0" />
    </CmdI>
    <CmdI id="GetStatus" direction="in" />
    <CmdI id="Status" direction="out">
      <PI id="Alarm" type="text/plain" dataRef="0" />
      <PI id="AC" type="text/plain" dataRef="0" />
      <PI id="Pumps" type="text/plain" dataRef="0" />
      <PI id="BatteryLevel" type="text/plain" dataRef="0" />
    </CmdI>
    <CmdI id="GetStatusPump" direction="in">
      <PI id="Serial" type="text/plain" dataRef="0" />
    </CmdI>
    <CmdI id="StatusPump" direction="out">
      <PI id="Model" type="text/plain" dataRef="0" />
      <PI id="ServiceDate" type="text/plain" dataRef="0" />
      <PI id="Alarm" type="text/plain" dataRef="0" />
      <PI id="Infusing" type="text/plain" dataRef="0" />
      <PI id="Serial" type="text/plain" dataRef="0" />
    </CmdI>
    <CmdI id="GetInfusion" direction="in">
```

```
27        <PI id="Serial" type="text/plain" dataRef="0" />
28      </CmdI>
29      <CmdI id="Infusion" direction="out">
30        <PI id="Rate" type="text/plain" dataRef="0" />
31        <PI id="Infused" type="text/plain" dataRef="0" />
32        <PI id="VTBI" type="text/plain" dataRef="0" />
33        <PI id="TimeRemaining" type="text/plain" dataRef="0" />
34        <PI id="Drug" type="text/plain" dataRef="0" />
35        <PI id="Weight" type="text/plain" dataRef="0" />
36        <PI id="Gender" type="text/plain" dataRef="0" />
37        <PI id="Age" type="text/plain" dataRef="0" />
38        <PI id="Height" type="text/plain" dataRef="0" />
39        <PI id="Serial" type="text/plain" dataRef="0" />
40      </CmdI>
41      <CmdI id="AlarmCommand" direction="out">
42        <PI id="Serial" type="text/plain" dataRef="0" />
43        <PI id="Message" type="text/plain" dataRef="0" />
44      </CmdI>
45    </SD>
```

Listing 5.3. Portal service interface

5.3.1 Status information

The workstation status information can be retrieved at any time via the PalCom interface. There are four different commands that can be used to retrieve status information.

- **GetIdentity**
  Used to identify a workstation. This command will respond with serial hospital, ward, bed and serial number of the workstation.

- **GetStatus**
  Will tell if the power is plugged in or not, the battery level of the workstation, an XML formatted list containing the serial numbers of all connected pumps.

- **GetStatusPump**
  Will respond with status information about a specific pump, identified by a serial number. The response will contain the model, the last service date, any potential alarm, the serial number and if it is currently infusing or not.

- **GetInfusion**
  Will respond with information about the patient, such as weight, length and gender. The respond will also contain what drug is infused, how much has been and will be infused as well as the infusion rate.

All these commands use polling to retrieve information from the workstation. When the command is invoked a new network connection will be established to the workstation. An XSLT request will be issued on the established connection. The XML response is parsed and delivered to the output command in question.

5.3.2 Alarm

The alarm functionality will set up a permanent connection to the workstation when the Portal software is executed. The permanent connection will reduce network load since data will only be sent by the workstation whenever something has changed. The Portal software will parse the received XML data and may invoke an alarm command if there is any critical value.

An issued alarm command will contain the serial number of the device triggering the alarm or none if the alarm was triggered by the workstation itself. An alarm is triggered on the workstation if the battery level is low or if the AC is unplugged. An alarm is triggered on a device if for example an ongoing infusion is coming to an end.

5.4 Conclusions

This situation shows how a PalCom interface can be developed without knowing exactly how it is going to be used. The alarm system present in Figure 5.1 is currently unknown and will probably be developed in another thesis in the future.

Obviously Alaris Health has thought about the problem of interoperability and for that reason they provide the XSLT interface. The XSLT interface is very complicated in comparison to the PalCom interface developed. To use the XSLT interface a software developer will have to be available while the PalCom interface can be used by someone without any software development skills.

Chapter 6

Software distribution

The purpose of this chapter is to describe the software distribution to anyone who wants to extend or modify the software. The content of the distribution package is described in Section 6.1. Section 6.2 describes how to build the software and Section 6.3 describes how to add support for additional pumps.

6.1 Distribution package

The latest distribution build is currently available at `http://mt.cs.lth.se/files/` (this may change and if so, it can be requested with an email to the author of this document).

The distribution contains all the software that has been mentioned in this report. Both precompiled, ready to use, jar files as well as the entire source code and build scripts.

The distribution is structured as follows.

```
1 /transmitted
2 /logs
3 /src
4 /bin
5 /logextractor.bat
6 /storage.csv
7 /run.bat
8 /portal.bat
```

Listing 6.1. Content of the distribution package

The *transmitted* and *logs* folders are empty by default. The purpose of these folders are to provide a location for transmitted (parsed) logs as well as unparsed logs. The *src* folder contains all source code and can be deleted since it does not provide any functionality. The *bin* folder contains all precompiled jar files.

6.2 Build process

There is an ANT (Another Neat Tool) script available in the root of the *src* directory. By default, this script will build all applications and generate a distribution file like "logalyzerYYYYMMDDHHMM.zip". The script can also be issued with the target "clean" which will remove any generated files. The compilation will need JDK 1.6 SE.

6.3 Implementing support for additional pumps

To add support for a new device an algorithm will need to be associated with the device. Without an algorithm the Parser service will not know how to interpret the content of the device log.

There are several possible approaches to develop and introduce new parser algorithms to the system. This section will describe one of them, using the Eclipse IDE.

The parser.jar file, as can be found in the *bin* directory of the distribution, will need to be present.

1. Create a new Java project in Eclipse.
2. Add parser.jar to the build path.
3. Create a new class with any name you like. Let the class extend ParserAlgorithm found in the parser.algorithm. package.
4. Implement the parsing algorithm in "public void parse()".

Every parser algorithm will need to be packaged in a JAR file. The manifest of the JAR file will need an attribute determining the path to the class implementing the algorithm. The attribute name is "Algorithm-Class" and the value should be set to the class name including package. The JAR file will need to be copied to the class-path of the device running the parser service.

The final step is to add information about the algorithm in the database. Each device model has its own row in the table called "models". The "parser" column will need to set to the name of the jar file, excluding the file ending.

Any class extending ParserAlgorithm will be served with a few helpful methods. These methods will provide the raw content of the log file along with methods for creating and storing events. For a detailed description of the methods, see the JavaDoc in the ParserAlgorithm.java found in *src/parser/algorithm/ParserAlgorithm.java*.

Chapter 7

Conclusions

This chapter will conclude and summarize the thesis. Section 7.1 will present a summary of the work. There has been some unexpected issues with working in a health care environment, this topic is discussed in Section 7.2. As this thesis has been performed close to the developers of PalCom it has had impact on the project. This topic is discussed in Section 7.4. Modifications of the PalBox software is discussed in Section 7.3. The differences between using a traditional approach insted of PalCom is discussed in Section 7.5. Section 7.6 will discuss potential future work.

7.1 Introduction

A real scenario involving pervasive computing has been investigated. Different networking technologies has been looked into and conclusions have been made. Palpable computing has been chosen as foundation of prototype systems aiming at solving the problems. The PalCom framework has been used and evaluated.

7.2 Working in a health care environment

In the health care environment, where this thesis has been carried out, it is easy to realize the potential chaos caused by pervasive computing. This thesis has focused on just a small part of that problem. This section briefly discusses how the wariness of this environment has affected the work.

A problem that has arisen is the lack of helpfulness from health care companies. While this may seem superfluous it is actually very easy to understand. The companies seem to be used to situations where peoples lives depend on there equipment. There is absolutely no room for mistakes. To answer a students emails just seems like an unnecessary risk. This attitude we believe will make it harder to introduce more flexible software solutions in the health care sector.

A few weeks after this thesis was initiated, one of the manufacturers released new software for their pump devices. This software requires a MSSQL database

to be present and the software transmits the log from the device and populate the database. Another software program analyses the content of the database and presents statistics, much like the statistics in our web application. Documentation of the database was requested. The company was not willing to give any support at all, even the tables in the database was said to be "secret". This is another example of the attitude in the health care sector as mentioned above. This software also shows an effort of satisfying the need for statistics on usage. But no effort is put into actually merging statistics.

7.3 PalBox modifications

PalBox is an application found in the PalCom distribution. It provides the functionality of running assemblies and unbound services. The PalBox has been used for both these purposes in the prototype system. The PalBox has been modified to make it more user friendly and make it more suitable for this specific situation.

By default the PalBox will look for assemblies and unbound services in the operating system users home directory. To avoid having to give descriptions on how to copy specific files to that folder, or doing it automatically resulting in a more complex un-installation process, this folder is changed. The modified PalBox will look for assemblies and unbound services in the current running directory of the application.

By default the PalBox will ask for version number when an assembly is loaded and instance when an unbound service is loaded. This behavior has been changed to simply always use version 1.0 and instance 1. The assembly and unbound services used in the distribution does not use this functionality. There were also buttons for unloading all unbound services and unloading all assemblies which have been removed. This modification will ease the use by decreasing the number of decisions taken by the user.

When constructing an assembly the DeviceID:s will be hard coded into the assembly description. This caused problems with the DeviceID of the PalBox as it changes depending on target system. The DeviceID is normally automatically generated from the MAC address of one of the Ethernet cards on the target system. This problem was solved by entering a known string ("palboxdevice") instead of the actual DeviceID in the assembly description. The assembly description will be parsed when loaded in the PalBox and the string replaced with the correct DeviceID of the current system.

7.4 PalCom evaluation

PalCom is an ongoing research project. It is a good idea to use scenarios like those dealt with in this thesis and find weaknesses in order to improve the quality. This section presents a few identified issues with PalCom that we believe needs to be improved.

When creating a command in a PalCom service, the name of the command can not be the same as any of the parameters. This will result in a class cast exception. When an assembly is created the name of any of the commands can not be same as the name of any of the parameters in any of the involved commands. This will also result in class cast exception.

We can see that if services are created independently of each other and then put together there may be problems with concurrency. We can see how there may be a need for unbound services to be created just to glue different services together. The reason being that the PalCom assembly does not provide any concurrency control. If the parser service developed in this thesis did not have the pre-load functionality, such a glue component would have to be developed.

The raw log files sent to the parser service have sometimes been a bit bigger then 1 mb. When files of this magnitude is sent via PalCom there will be warning messages about "Trailing ACKS". This situation was handled by compressing the data being sent. The maximum compressed size is now about 150 kb. All communication between services has been carried out using UDP. UDP may sometimes have a upper packet size of 8 kb. There is an option that can be enabled in the services that would make the transmission reliable. PalCom will make sure the message is chopped up into small enough peaces that will be delivered correctly. We suggest that this reliable option is turned on by default.

PalCom uses its own threads (see Section 3.5). These `PalcomThreads` are a bit hard to work with. The experience from the development part of this thesis is that the probability of starvation is high. We believe that the implementation of services in an unconstrained environment (a modern PC with almost unlimited resources) can be made easier. A new class extending the existing `AbstractService`, or replacing it, could be developed with the intention to be used in such environments. That class would be based on Java threads. We have also seen how the amount of code in the implementation of `AbstractService` tend to grow. The creation of commands, and adding parameters to these commands, tends to result in many lines of code. It should be possible to reduce the amount by implementing helper methods in, i.e, the `AbstractService` class.

We found that it was hard to distribute the software since the device id in the assembly description was hard coded. This was dealt with by parsing the assembly description when it is loaded into the modified PalBox. After discussions with the developers of PalCom there is now another solution to this problem. When creating an assembly description a device id can be set to be dynamic and correspond to the device id of the device running the assembly.

A potential problem with the heartbeat mechanism used by PalCom (see Section 3.5) is large networks. By default the heartbeats are sent every other second. In a large network and/or with many services running there may be alot of heartbeat messages sent resulting in reduced network performance. This problem has however not occurred in our prototype system.

When there is a need to retrieve a list of values from a service, XML has been used. If one service provides a list of values (in XML) and another service requires

those values (also formatted as XML) those parameters can till not be safely combined. The XML may express the values using different notations of elements and attributes. We believe that if there was a list data type available in the service descriptions, alot of these situations would be easier to deal with. The obvious drawback of adding new functionality to the services is that it may grow to become to advanced. This is a hard dilemma to deal with since the goal of PalCom has been that it should be easy to combine services.

ZeroConf (see Section 2.3.2) uses a service id to determine the type of a service. We believe that such a service id may have to be introduced in Palpable computing. As mentioned in Section 4.1, one early consideration was to make each pump palpable. With the current architecture of palpable computing this will require one assembly per pump which is virtually impossible with 700 different pumps. If there was a service id mechanism identifying each pump as a pump then a more dynamic assembly could be developed.

7.5 PalCom versus a traditional approach

The end user of the prototype system described in Chapter 4 may ask why he or she has to run two programs simultaneously. The prototype has been developed with the intention that it should by used in a larger scale. The parser service should be able to parse log data from many different kinds of devices. The storing service should be able to provide persistent storage to more service then just the parsing service.

It is easy to fool a customer into thinking there is a greater value in the product then there actually is. If the customer is introduced to a software system that only requires one program to run, he or she will probably choose that one instead of the one requiring two programs to run. But the software system requiring only one program will introduce just another very specialized solution making the global IT infrastructure more complex. While the software system requiring two programs may reduce complexity in the long run.

The system presented in Chapter 4 could have been developed as one large Java application. The parser and storage services would probably have been as packages in that application. This approach would work in this small scenario but problems arise when it has to be extended. If, instead of creating plugins in a traditional approach, we create PalCom services, and let them be visible on the network, we can create lightweight user interfaces to full fill user requirements. The user interfaces will themselves be services, like the Service station in Chapter 4. Instead of distributing executables, assemblies are used.

The online surveillance scenario in Chapter 5 shows another advantage of PalCom. The XSLT interface is very complicated in comparison with the developed PalCom interface. While a software developer will have to be available to use the XSLT interface, anyone without any software development experience can interact with the PalCom interface, i.e via a PalCom browser, and use it in assemblies.

7.6 Future work

The online surveillance, as described in Section 5, can be further developed. There has been work done on making PalCom run on cell phones. To make the data available by the online surveillance service available on a hand held device would be interesting. For example the cell phone could start to vibrate whenever there is an alarm and also show the ward and bed triggering the alarm.

Appendix A

Installation manual

The purpose of this chapter is to describe how to install the software. The installation process of the web application is described in Section A.1. The installation process of the PalCom services are described in Section A.2.

First of all the distribution package is needed, direction on where to find the distribution can be found in Section 6.1. The distribution package can also be requested by email (see first page). Unpack the distribution at any location you like.

A.1 Web application

This section will tell you how to setup the web application from scratch. All files needed to set up the web application is available in src/projects/www. The application has been tested on the following configurations:

- Apache 2.2.9
- PHP 5.2.6
- PHP-GD 2.0
- MySQL 5.0.51

First of all you need a target machine running a web server capable of executing PHP and having PHP-GD enabled. You also need a database server running. Since Apache 2 and MySQL has been used during development they are set as requirements even though you may be able to run the system with another configuration.

You need to add a database and a user to interact with it. Here is an example of how to create a database (logalyzer), a user (logalyzer) with a password (dummypass) and grant full administrative privileges to the database.

```
1 create database logalyzer;
2 grant usage on *.* to logalyzer@localhost identified by 'dummypass';
3 grant all privileges on logalyzer.* to logalyzer@localhost;
```

Listing A.1. Create and add a user to MySQL

The next step is to add required tables to the database. There is a SQL script available in www folder that will create all necessary tables. The script is called logalyzer.sql. The script can be executed as follows.

```
1 mysql logalyzer -u logalyzer -pdummypass | `cat logalyzer.sql`
```

Listing A.2. Execute SQL script in MySQL

Next we configure the web application to connect to that database. The database configuration is found in logalyzer/includes/database.inc.php. The attributes to edit are found at the very top of the file and looks like follows.

```
1 mysql_connect("localhost","logalyzer","dummypass");
2 mysql_select_db("logalyzer");
```

Listing A.3. Set up database connection

mysql_connect[1] will need the host name or IP-number where the MySQL server is running, as well as user name and password. mysql_select_db[2] will need the name of the database to use.

The web application is now ready to be used. Copy the files located in www/logalyzer to a location within the web server root folder. There is a script located at logalyzer/database.php that will handle all database communication. You will need to know the actual URL of this script in order to set up the PalCom services correctly.

A.2 PalCom services

This section will tell you how to set up and run the PalCom services. There are three services involved and two of them are unbounded (see Section 3.4.2) and will need a PalCom runtime environment. An application called PalBox (see Figure A.1) is used to provide this runtime environment.

The software described here has been tested on the following configuration.

- Windows XP SP2 and Vista SP1
- Sun Java Runtime Environment 1.6.0

All files needed to set up the PalCom services can be found in the root folder of the distribution.

The first thing to do is to configure the storage service. This is done by creating a file called storage.csv in the root folder. If this step is not performed, this file will be automatically created with the following content.

[1] http://se.php.net/manual/en/function.mysql-connect.php
[2] http://se.php.net/manual/en/function.mysql-select-db.php

```
1 mode|http
2 ip|mt.cs.lth.se
3 database|/database.php
4 port|80
5 username|logalyzer
6 password|dummypass
```

Listing A.4. Storage service configuration file content

Your file should follow the same syntax, attributes are followed by a pipe and terminated by a new line. In this example the URL of the database.php script is http://mt.cs.lth.se/database.php.

- **mode** - Should always be "http". This attributes determines how to store the information received. It provides forward compatibility, in the future there may be other ways of communicating and this attribute will make sure old configurations are compatible.
- **ip** - Should be the host name or IP-number part of the URL.
- **port** - Should be sett to the IP port where the web server is running.
- **username / password** - Should be set to the user name and password used when setting up the web application.

The next step is to configure the PalBox to load the assembly and unbound services at startup. Run the run.bat program located in the root folder of the distribution. Two applications will appear, the PalBox (see Figure A.1) and the Service station (see Figure B.2).

Figure A.1. The PalBox device.

The assembly and the services are located in the bin folder in the root of the distribution. Click on "Load Unbound Service" in the PalBox GUI. A new window with a drop down menu will appear, as in Figure A.2. There is two alternatives in the menu, choose on of them and click "Ok", then choose the other one and click "Ok". Next click on "Load Assembly". Browse to the bin folder and choose "Unbounded.ass", then click "Ok" (see Figure A.3). There are now two services

running as well as an assembly. All configurations are saved and will be automatically loaded when starting the PalBox in the future. The settings are stored in "storage.csv" in the root folder.

Figure A.2. Unbound service in the PalBox device.

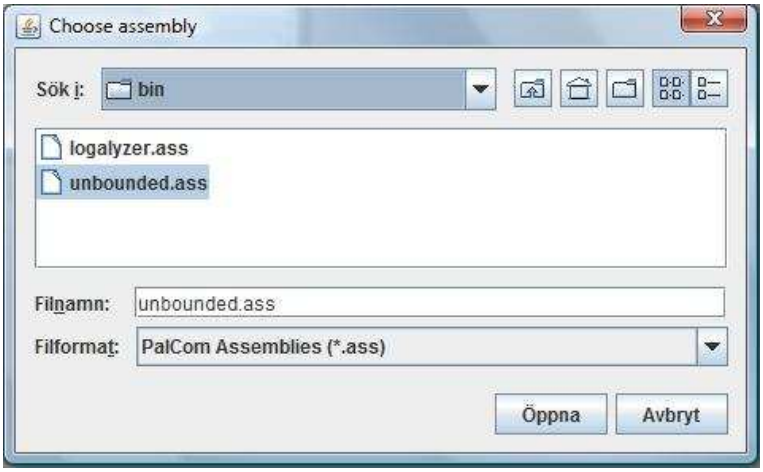

Figure A.3. Assembly in the PalBox device.

The service station will look for log files in a specified folder. Transmitted files can optionally be moved to another folder when successfully transmitted. These folders can be set up by clicking on the browse buttons in the Service station. Any changes will be recorded and will be kept between usage sessions. The settings are stored in "settings.csv" in the root folder.

Appendix B

User manual

This is a user manual for the prototype system (as presented in Chapter 4). It is written with the intention that anyone with only limited computer skills should be able to understand it.

B.1 Downloading log data

The following pump devices are supported through vendor software:

- Cardinal Health, Alaris GP
- B. Braun, Infusomat Space
- B. Braun, Perfusor Space
- Alaris IVAC Signature Gold Edition, 7231AX
- Alaris IVAC Signature Gold Edition, 7131AX

The LogExtractor program is found in the root of the distribution package. It can be executed using the logextractor.bat file found in the root of the distribution package. It is developed only for the Alaris CC Guardrails, manufactured by Cardinal Health.

When LogExtractor is loaded a user interface will appear showing a list of COM ports available on the computer (see Figure B.1). Each com port is described with a button labeled with the COM port number as well as a status text. The status text may be "Not connected" if there is no pump connected to it, or it may be "Connected to X" where X is the serial number of the connected pump device.

To download log data, press the button corresponding to the pump device. The software will start downloading. A progressbar will be continuously updated as well as the status text. The status text will contain the number of events waiting to be transmitted. When the downloading progress is complete a new window will appear asking where to store the downloaded data.

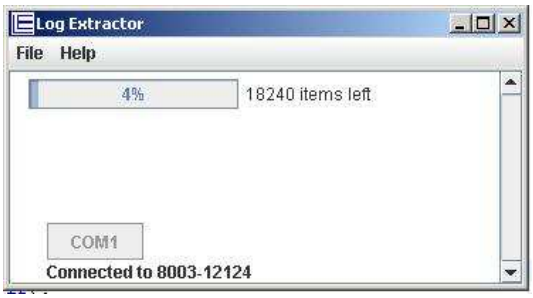

Figure B.1. Download software - main GUI.

B.2 Service station

This section will describe the functionality in the Service station. It can be executed using the run.bat file found in the root of the distribution package.

The File menu simply provides a way of exiting the application, this can also be done by pressing the X at the top right corner of the application. The Help menu provides an "About" item with information about the developer of the application.

There is a panel at the top of the application with configuration and general interaction functionality. Beneath the panel, interaction specific to an individual device is present.

There are two folders that may be changed to serve the needs of the end user. One folder containing new log files, "Unparser logs", and one containing transmitted logs, "Parsed logs". These folders are changed by clicking on the respective browse button. The "Select all" button will select all devices available. When clicked it will change to "Unselect all" and provide the functionality of unselecting all devices. The "Send selected" button will send the selected devices, meaning the log files will be parsed and made available in the statistics provided by the web application. The "Clear sent items" will move the successfully parsed log files to the folder specified in "Parsed logs".

Each device panel will provide information specific to that device. The device panels are separated into groups where each group is the content of the same folder on the file system. The groups are separated by a label with the file system path of the folder. The unique inventory number associated with the device are present on the border of the panel as well as the file size of the log. A check-box, determining if the device is "selected", as well as the filename of the log is located to the left. A "Send" button, providing the functionality of sending that specific log, is present to the right. A status field is located in the center.

The screen shot supplied in Figure B.2 show a situation where three devices has been selected and three devices has been sent. Two of the files has been successfully parsed and has been made available in the statistics. The third device has encountered an error. This screen shot shows how the status label can be used. Here the

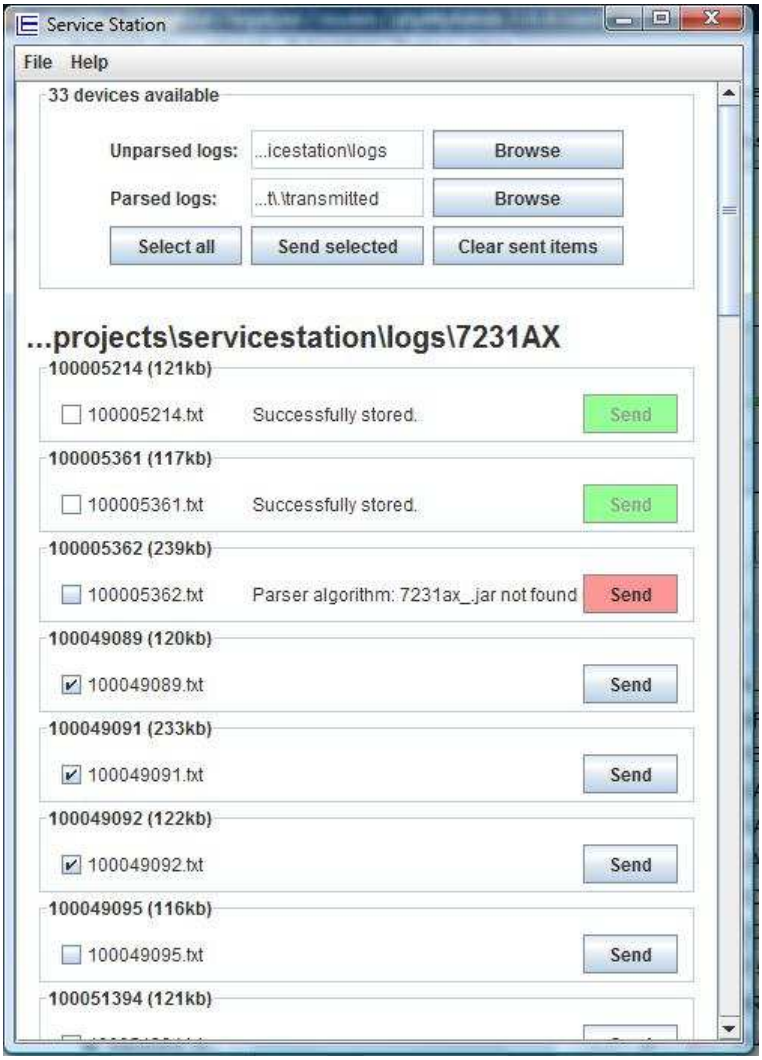

Figure B.2. Service station GUI.

database is miss-configured and will respond with a parser algorithm that is not available. The parser service will look for the algorithm without success, leading to an error response to the Service station. The "Send" button will turn green if the sending procedure was successful and red if unsuccessful.

B.3 Web application

The web application provides statistics on usage. Figure B.3 shows the main menu and filtering options. The statistics page shows the filtering options. The admin page provides administration functionality.

Figure B.3. Web application.

The admin page is used to update the placement of the pumps and add new pump devices to the database. A CSV file formatted as Listing B.1 can be uploaded and will update the database.

```
1 Inventarie;Kund;Modell;
```

Listing B.1. CSV format of uploaded placement information

The application is based on the principle of filtering data on different abstraction levels. At any level, as much information as possible is presented. When a division is selected, only data from that specific division is used and so on. The available filters are division, year, month and day.

When selecting a specific division, drug usage (see Figure B.4) and associated device models (see Figure B.5) are presented.

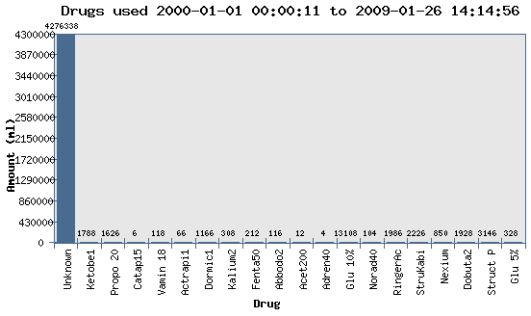

Figure B.4. Web application - drug usage.

When selecting a year, the same statistics are presented but now based only on that specific year. The same goes for selecting a month and a specific day. When selecting month, two new graphs appear. One presenting the number of different devices used each day during that month (see Figure B.6), and one presenting the maximum number of simultaneously used pumps during that same period (see Figure B.7).

When selecting a specific day, a time line will be presented, see Figure B.8. The time line will show the usage of each pump during that day. Each event is presented by a blue line. When clicking on that blue line all information about that event will be presented as in Figure B.9.

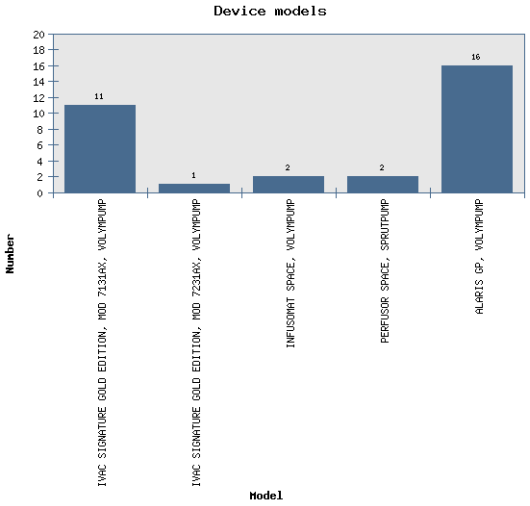

Figure B.5. Web application - device models.

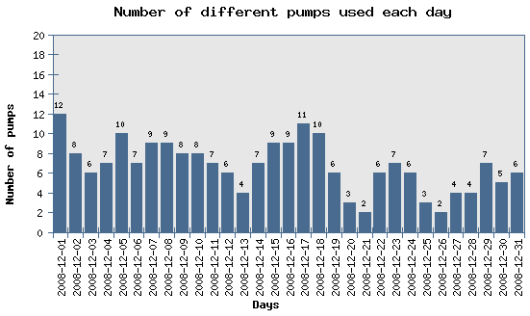

Figure B.6. Web application - number of different devices.

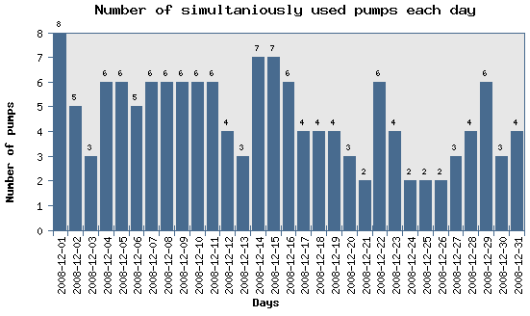

Figure B.7. Web application - maximum number of simultaneously used devices.

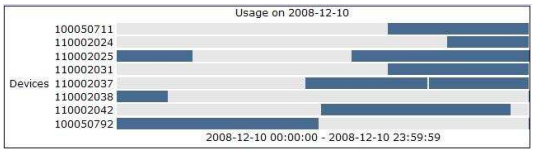

Figure B.8. Web application - time line.

Pump operation event for pump 100050786.

Start	2008-12-29 01:57:39
End	2008-12-31 00:09:54
Amount	602.62ml
Druglib	ANIVA (v 1)
Syringe	BD Plastipak 50ml
Drug	Propo 20

Figure B.9. Web application - pump event.

Appendix C

Portal online surveillance manual

The purpose of this chapter is to describe how to set up and use the Portal software, described in Chapter 5.

C.1 Preparing the PC

The PC will have to be configured with the IP address 192.168.0.130 and the netmask 255.255.255.0. When this is done the PC should be able to connect to the workstation via any web browser. The workstation should be available on 192.168.0.128. There is a web server running on port 80 and an XSLT server running on port 5020.

C.2 Configuring the Portal software

The address of the workstation and the port of the XSLT server is configurable. The configuration is done by creating a file called portal.csv with the content in Listing C.1.

```
1 port|5020
2 host|192.168.0.128
```

Listing C.1. Portal configuration file

C.3 Using the Portal software

To run the surveillance service the portal.bat file, found in the root of the distribution package, is executed.

Any PalCom browser can be used to monitor the service. The Eclipse Browser has been used during development.

Bibliography

[1] Eric van der Vlist *XML Schema, The W3C's Object-Oriented Descriptions for XML* O'Reilly, June 2002

[2] XSL Transformations (XSLT), 2009-04-23 (13:08); http://www.w3.org/TR/xslt

[3] AvGeorge F. Coulouris, Jean Dollimore, Tim Kindberg *Distributed Systems: Concepts and Design* Pearson Education, 2005

[4] Mark Weiser, "The Computer for the 21st Century" *Scientific American* 265(3):94-104, September 1991

[5] David Svensson Fors, Boris Magnusson, Sven Gestegård Robertz, Görel Hedin, Emma Nilsson-Nyman. "Ad-hoc Composition of Pervasive Services in the PalCom Architecture." In Proceedings of International Conference on Pervasive Services *ICPS09, ACM, London July 2009*

[6] Wikipedia, the free encyclopedia, "Interoperability", 2009-02-26 (13:30); http://en.wikipedia.org/wiki/Interoperability

[7] Wikipedia, the free encyclopedia, "Plug and play", 2009-02-26 (13:30); http://en.wikipedia.org/wiki/plug_and_play

[8] Intel, "Continua", 2009-02-26 (14:54); http://www.intel.com/healthcare/continua.htm

[9] Grace A. Lewis, Edwin Morris, Soumya Simanta, Lutz Wrage, "Why Standards Are Not Enough to Guarantee End-to-End Interoperability" *Seventh International Conference on Composition-Based Software Systems* 164-173, 2008

[10] Da Ronco, S.; Diez-Andino Sancho, G.; Dini, L.; Ronchieri, E.; Selmi, M. "Integrated Solutions for Information Sharing In Health Care Applications" *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)* 254-254, 2008

[11] Carroll, R.; Cnossen, R.; Schnell, M.; Simons, D. "Continua: An Interoperable Personal Healthcare Ecosystem" *IEEE Pervasive Computing* Issn: 15361268, Volume: 6, Issue: 4, Pages: 90-94, Year: 2007

[12] Lopez, D.M.; Blobel, B.G.M.E. "A development framework for semantically interoperable health information systems" *International Journal of Medical Informatics* volume: 78, issue: 2,83-103, 2009

[13] Jan Newmarch *Foundations of JINI2 Programming* Apress, 2006

[14] Jini.org. "Jini Specifications" 2009-02-26 (14:54);
http://www.jini.org/wiki/Category:Jini_Specifications

[15] Wikipedia, the free encyclopedia, "DLNA" 2009-02-28 (10:15);
http://en.wikipedia.org/wiki/Dlna

[16] UPNP Forum 2009-02-28 (11:34);
http://www.upnp.org/

[17] DNS SRV (RFC 2782), "Service Types" 2009-02-20 (10:55);
http://www.dns-sd.org/ServiceTypes.html

[18] Wikipedia, the free encyclopedia,"Zero configuration networking" 2009-02-29 (15:43);
http://en.wikipedia.org/wiki/Zero-configuration

[19] David Svensson Fors "Assemblies of Pervasive Services" *Lund Institute of Technology* 2009

[20] Brice Jaglin, Boel Mattsson. "Implementing the PalCom protocol in an Axis network camera" *Lund Institute of Technology* January 2007

[21] Thomas Forsström, David Raimosson. "External ad-hoc communication for PalCom" *Lund Institute of Technology* January 2007

[22] PARC, "Palo Alto Research Center " 2009-03-01 (17:07);
http://www.parc.com/

[23] The Obje Software Architecture, "Obje Interoperability Framework" 2009-03-01 (13:49);
http://www.parc.com/research/projects/obje/Obje_Whitepaper.pdf

[24] PalCom web site. Palpable Computing - a new perspective on Ambient Computing 2009-03-05 (08:45);
http://www.ist-palcom.org/

[25] Wikipedia, the free encyclopedia,"Bluetooth", 2009-03-14 (12:21);
http://sv.wikipedia.org/wiki/Bluetooth

[26] Bluetooth, "About the Bluetooth SIG", 2009-03-14 (12:21);
http://www.bluetooth.com/Bluetooth/SIG/

[27] Sun Microsystems 2009-03-14 (12:21);
http://www.sun.com/

[28] Michael Anthony Blackstock "A Common Model for Ubiquitous Computing" *The University Of British Columbia* October 2008